Parameters for the Evaluation for the Choice of a Good Programming Language

Baah, Barida
Department of Computer Science
University of Port Harcourt
Port Harcourt.
Rivers State, Nigeria.
baridakara1@yahoo.com

Taylor, Onate Egerton
Department of Maths/Computer Science
Rivers State University of Science and Technology
Port Harcourt.
Rivers State, Nigeria.
tayonate@yahoo.com

***Abstract*— Most of the problems face today by business, organization, industrial and companies that uses automated data processing system is as a result of wrong choice in considering a programming language for an effective and efficient system implementation because of this wrong choice of language most business, organization and industrial fails to achieve it set goals. This research papers will take a critical x-ray of those parameters that will be consider before choosing a particular language in other to achieve a good industrial, organizational and business set goals. In this research paper work we shall look at flexibility of the language, Interface design of the language, Speed of the Language, Accessibility of the Language, Compatibility of the languages, Language Safety, Responsiveness of the Language and Expressiveness of the Language. Finally in this research paper work we shall also develop a simple algorithm and a flowchart that will test these parameters by using assign variable to depict this various parameters, to test to ascertain whether a particular programming languages meets these parameters for evaluation or not. Finally, the methodology that is used in this research is called Structural System Analysis and Design Method (SSADM) and it was implemented using Turbo Pascal programming language version 1.5.**

***Keywords*— Parameters, Evaluation, Accessibility, x-ray, Responsiveness, flexibility, Expressiveness, Interface design, Completeness and Compatibility.**

I. INTRODUCTION

Every day we as human being are face with the problem of choice in our day to day life the choice we make today definitely affect our life as an individual either in a positive way or in a negative way. So the choice we make has a lot to do with us; that is to say that if you make a good choice as an individual you will have a positive or good result but when you make a wrong choice it will yield to negative results.

This is also applicable to any business, organization, industrial and any other Parastatals the day to day running of every businesses is wholly dependent on the choice or decision that is made. In business organization that require automated computer system in it implementation, the choice of the programming language has a lot do if such business, institution or organization must achieve it sets goals. So for any programming language to be chosen care must to taken to look at that programming language carefully before going for that language, because wrong choice of language will certainly make the organization to fail due to wrong choice of programming language.

II. Parameters for the Choice of a good Programming Language

(1) Flexibility of the Language: Flexibility of the programming languages has to do with the ability of the choice of a particular programming language to allow for changes without altering some part or the entire program.

(2) Interface design of the Language*:* This is another very important aspect to consider when choosing a particular programming language, the level of interface design define usability of the programming language. So the programming language should have a very good interface design as it will enable new programmers or professionals programmers to make good use of the language easily without spending much time to study how to use the language due its complexity in the interface design of the programming language.

(3) Completeness of the Language: We all need libraries in order to make reference to whenever we are face with difficulties, so also in choose of a good programming language it must have a good libraries system in order to assist new programmers or even professional programmers to make reference to. Assuming you want to start your next project writing XML parsers and DB integrations, or you just want to find the right library and get started? I try out a lot of languages, and this is where most of the otherwise great languages fall flat. This is a chicken and egg problem because how many libraries there are for your language is mostly a function of how many users you have, and you don't attract a lot of programmers when they need to write everything themselves from scratch. The best way to have instant access to lots of libraries is to have seamless C integration, since almost anything you will ever need was written in (or for) C. Or you could just grow a giant user-base like Perl or Python.

(4) Speed of the Language: There is a wise saying that "time is money" So a program that achieve great thing within a low speed is not better than a program that cannot achieve

great thing but it much more faster in it execution of program at a high speed. Speed appears to be the only criterion on some peoples list of features. But just because its over-valued doesn't mean its not important. The claims of which language is fastest is hotly contested, I will just say that some languages, like C, C++, Scheme, and OCaml pay close attention to runtime performance, and some (like Ruby) do not.

Most languages let you call C code as the escape valve for performance, but in todays computers speed comes from optimized memory layouts and cache efficiency, so to keep that C code fast, the data set needs to be already in memory in an efficient structure, which means that you will end up writing more and more C code to expose that data to your higher level language, and in the process importing most of the C resource management headaches as well. Languages that give you explicit control over memory layout or have seamless integration with C (like C++ or objective-C) have a clear performance advantage.

There is also a class of languages like OCaml and many Scheme and Lisp implementations that have very good performance most of the time, so a trip to C is rare, and the final note on speed is that often the best way to speed up code is to improve the algorithm. The advantage of some of the slower high-level languages is that implementing the sophisticated algorithms is much easier.

(5) Accessibility of the Language: In this case, in making your choice for good programming language care must be taking to know if the programming language is easily accessible or not but if the language is not very much accessible no matter how good that language is, there is no need to go for that particular programming language because whenever there is a problem with that language, may be due to corruption on your computer system or virus as the case may be and you need to install the language it will be difficult for you to get back the language and install on your system.

(6) Compatibility of the Language: The compatibility of a programming language is the ability of programming to be compatible to other programming languages so as to enhance the workability and the efficiency of the language in the realization of the business or organizational goals. It has to do with integrating a particular language to another programming language so as to have robust system implementation. For example integrating or combining MatLab with C/C++ program.

(7) Language Safety: Have you ever heard an OCaml or Haskell programmer claim that "once it compiles, it just works"? It is a bit of an exaggeration, but not as much as you might think. Often your compile errors are pointing out real issues you need to address for the correctness of your code, and sometimes they point out issues in your whole approach that will cause you to rewrite sections of your code *without ever having run the broken version*.

That is safety. It is the feeling that your programming language is watching your back. Haskell is a clear winner on the safety front. Its type system is powerful enough to specify some surprisingly sophisticated constraints. OCaml is a close second, and it falls off dramatically after that. It is also important also to point out that dynamic typing has a significant disadvantage when it comes to safety because you need to run the program to find the error, but since they usually do a pretty good job of trapping the error at runtime in ways that are easy to debug, they have an advantage over languages without any runtime support such as C or C++.

(8) Responsiveness of Language: In language responsiveness for example in Excel, when you change a number in a cell, all the other cells are updated immediately. That is responsiveness. The more steps between changing code and seeing the result of the change, the harder it becomes to stay in the programming flow.

The big winner in this area is Smalltalk with its live image. But other languages with powerful REPLs like Lisp and Scheme come in close second.

OCaml, Haskell, Ruby, and Python all have REPLs, but any Lisper will tell you that they are just not the same. Not that they aren't useful because you could be using any of the other languages where the best case scenario is fast compile times so you can get through your edit-compile-run cycle.

The reason why a good programming language needs to be responsive is so that you can be productive. And it is not just a matter of counting the wasted time waiting for compiles (although it can be significant), it is the destruction of your flow. When your changes are instant, you keep making changes and your mind settles into the contours of the problem. When you have even 1 minute breaks between changes, you lose track of what you were doing, and you lose focus.

(9) Expressiveness of the Language: The expressiveness of a programming language is the ability to reshape the language until you can express your program naturally. Some languages like Lisp and Scheme let you implement a new internal language for describing your program concisely. Smalltalk, Ruby and Haskell have basic building blocks and lightweight syntax that makes it easy to define new language constructs for your particular problem.

But it is not all about making embedded mini-languages, that is just one very effective version of expressiveness. It is also how well the language supports you with features that let you remove boilerplate code and just write the code that is needed to do the job.

It is a hard quality to pin down, but in many ways it is the most important because code is meant to be read by humans first and the computer second. Each line of code is a liability, it will need to be maintained. If it is not doing anything for you but telling the compiler things it could have figured out for itself, then it's a line wasted.

More than anything, this is the quality that attracts programmers and rewards them for their effort in learning the language. That is the reason that Ruby meta-programming took off after Rails hit it big: when people saw what could be done, how you could write working Ruby that read like a pseudo-code description of the program, they became drunk on the possibilities it opened up. Writing code in a highly expressive language is *fun*.

III. Research Methodology

The methodology that is beseech in this research work is called the structural system analysis and design method which is a waterfall method for the production of an information system design. SSADM can be thought to represent a pinnacle of the rigorous document-led approach to system design.

Structural System Analysis and Design Method (SSADM) is a systems approach to the analysis and design of information systems. one particular implementation of structural system analysis and design method which is builds on the work of different schools of structured analysis and development methods, such as Peter Checkland's, software system methodology, Larry Constantine's Structured Design, Edward Yourdon's Structured Method, Michael A Jackson's, Jackson Structured Programming and Tom DeMarco's Structured Analysis (Mike, 1999).

The reasons of chosen this research methodology is due to the following advantages:

1. The SSADM is mature
2. SSADM provide a clear separation of logical and physical aspects of the system.
3. It is well-defined techniques and also well documented.
4. It also provides an environment for the user involvement also.

IV. Algorithm for Parameters Evaluation

1: Display Message "Welcome to a Program to test for Parameters for "Evaluation of the Choice of a Good Programming Language"

2: Declare Choice as Character
Display Message "Enter Your Choice of Programming Language"
Choice

3: Display "Is the Programming language Flexible?"
Display "Do the language have good interface design?"
Display "Do the language have library (completeness)?"
Display "Is the Programming language Fast?"
Display "Is the Language easily accessible?"
Display "Is the Language Compatible?"
Display "Is the Language Safe?"
Display "Do the language update formulae immediately for change (Responsiveness)?"
Display "Is the language Expressiveness?"

4: Display "Enter *Y* for Yes and *N* for No"

5: Declare Array Parameter[10] as characters
Declare I and Yes as integer
Initialize Yes = 0

6: Setup Loop While I< 9
Test if Parameter[I] = "Y" OR Parameter[I]== "y" Then
Yes → Yes + 1
Test if Parameter[I] = "N or Parameter[I]= "n" then No → No + 1
EndLoop
EndIF

7: Test If Yes >= 6 Then
Display Choice, "is a Very Good Programming Language Choice"
EndIF

8: Test If Yes < =4 Then
Display Choice, "is a bad Programming Language Choice"
EndIF

9: Test If Yes == 5 Then
Display Choice,"an Average Programming Language Choice"
EndIF

V. A Program Flowchart

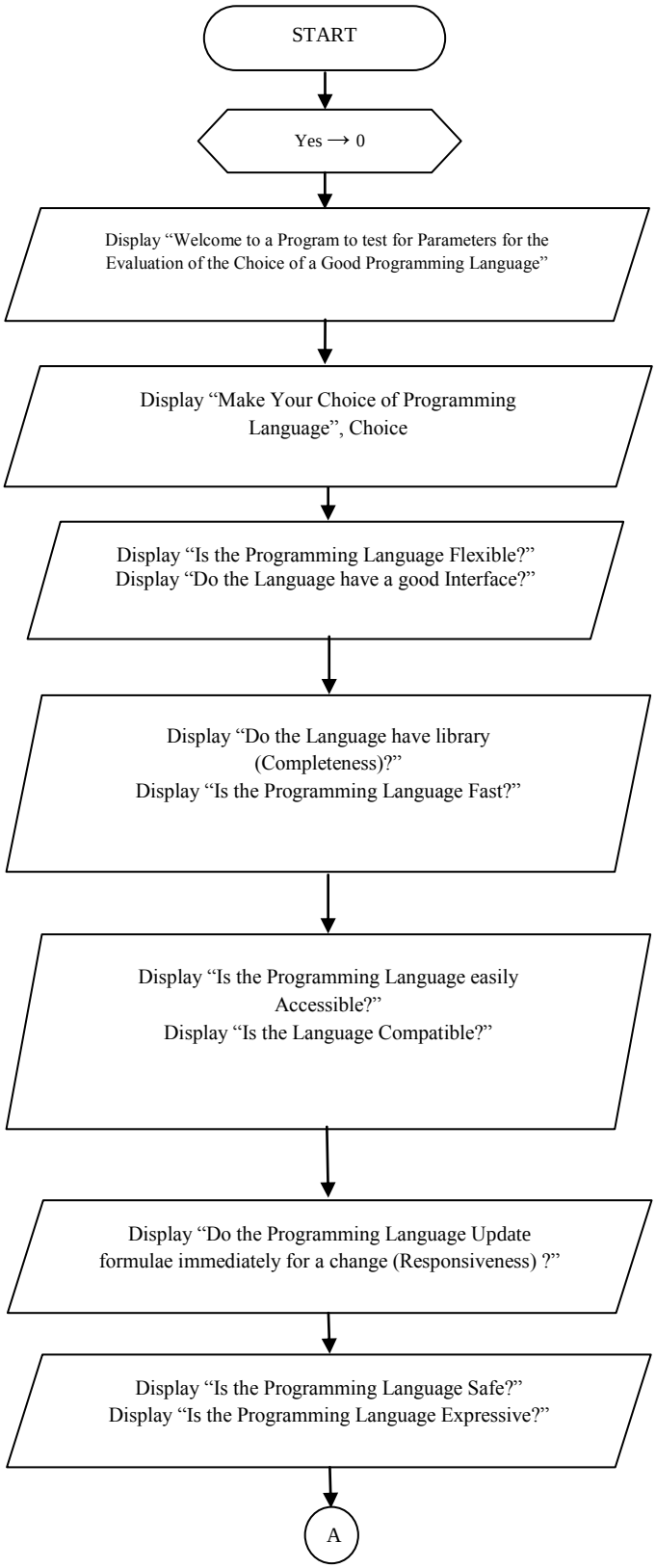

Fig 1 A flowchart showing a Welcome Message and a Display of Parameters for Evaluation

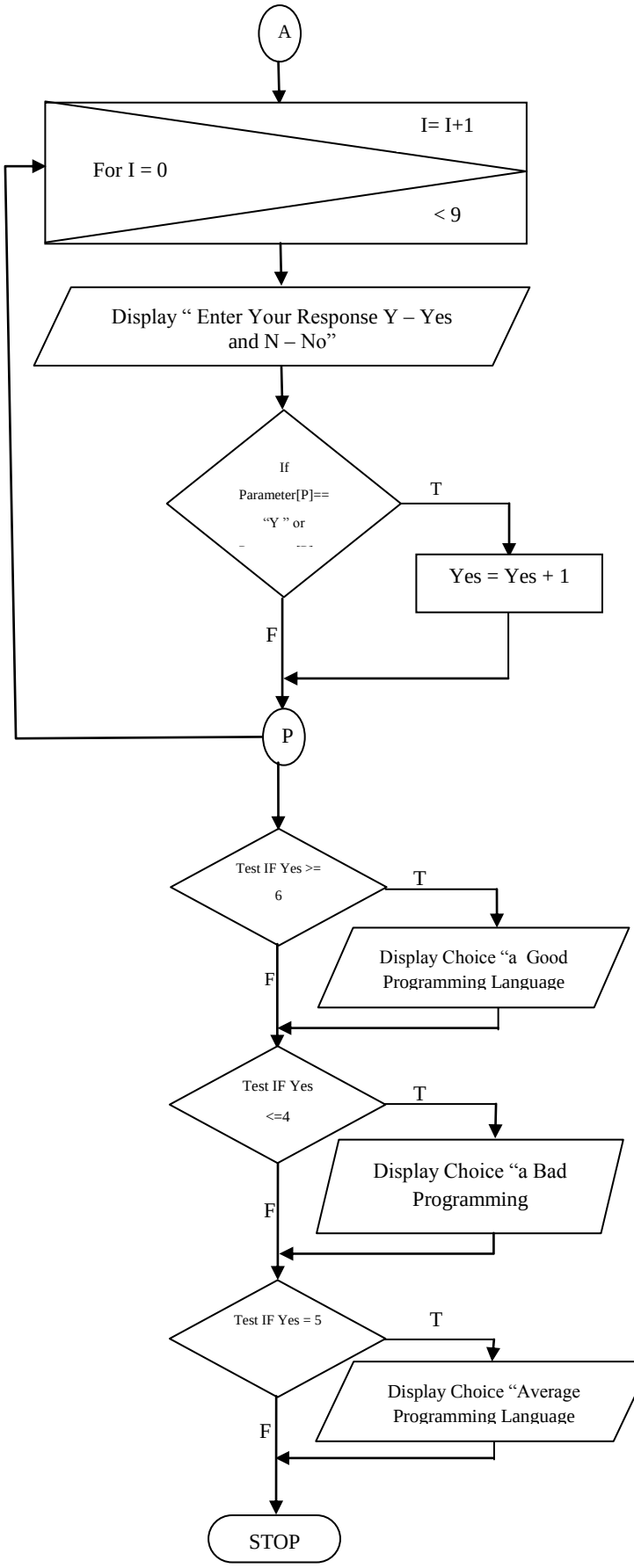

Fig 2 A flowchart showing Parameter test for choice of a Good Programming Language

VI. Input and Output Specifications

This addresses how data are been capture from the user through the keyboard by entering the data items and how the results are been display on the screen as it is shown in the figures below:

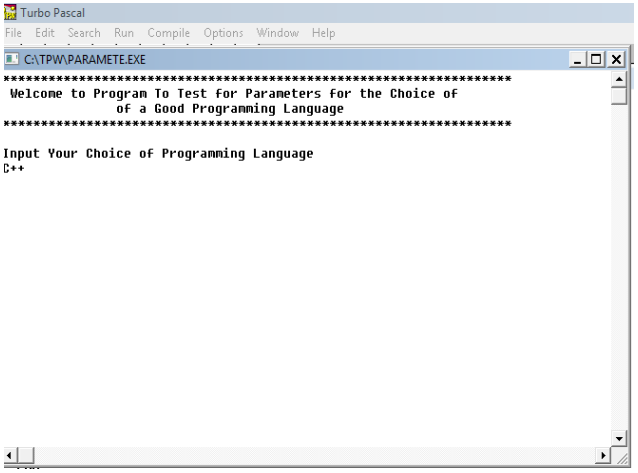

Fig. 3 A Screenshot showing a welcome message and "C++" input as choice of Programming Language

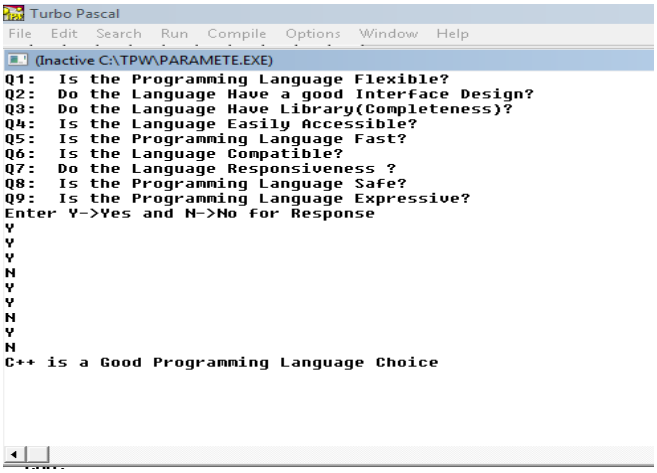

Fig. 4: A Screenshot showing Response and Result display after evaluating the parameter for choice of C++

From Fig. 4 above shows that C++ programming language is been evaluated as a good programming language choice based on the sets of questions which required a Y for Yes and N for No Responses; to shows that the programming language actually meet up a good choice of language.

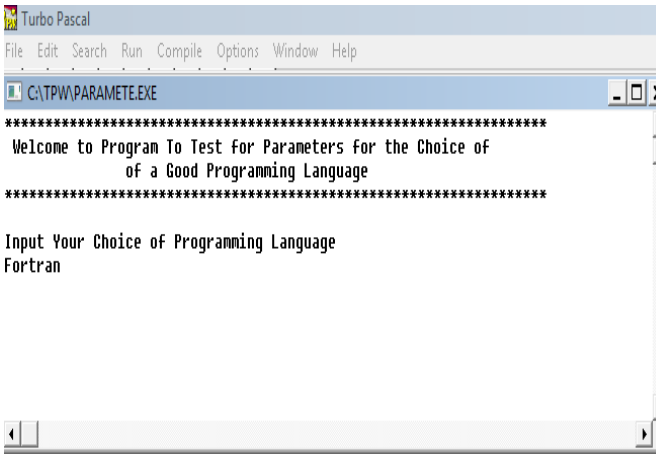

Fig. 5 A Screenshot showing a welcome message and "Fortran" input as choice of Programming Language

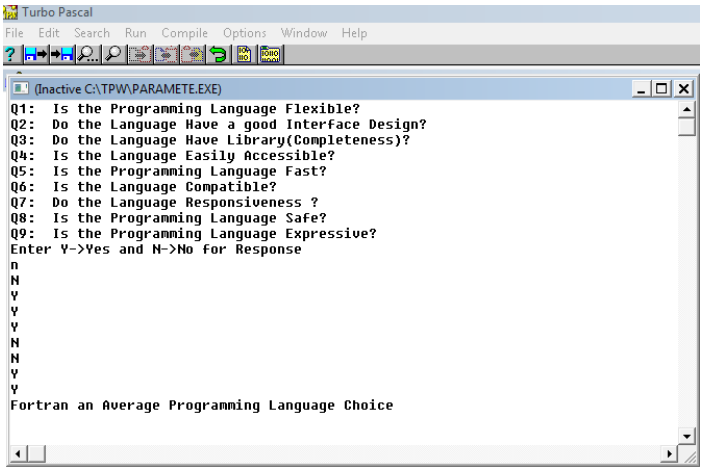

Fig. 6 A Screenshot showing Response and Result display after evaluating the parameter for choice of Fortran

Again, from Fig. 6 above shows that Fortran programming language is been evaluated as an average programming language choice based on the sets of questions which required a Y for Yes and N for No Responses; to shows that the programming language meet the average bench mark of choice.

VII. Conclusion

In a general conclusion, the Choice of a good Programming language depend on the parameters which is use for evaluation of the language, this is to say that the success of any business, parastatals, institutions and organizations is dependant on the choice of the language that is use in system implementation. Therefore, care must be taking in considering these various parameters that is use for the evaluation to check

whether most of the parameters are meet, before considering a particular programming language as good for business, parastatals, institutions and organizations because wrong choice of language is capable of ensuring that the set goals are not achievable thereby leading to general systems failures, due to wrong choice of programming language.

ACKNOWLEDGMENT

We commend the wonderful efforts of Causal Productions and also acknowledge the effort of Michael Shell and other contributors for developing and maintaining the IEEE LaTeX style files which we have use in the template format in the preparation of this research paper work.

REFERENCES

[1] Gupta V., (2007),Comdex.NET Programming. DreamTech Press, New Delhi, India. pp. 2

[2] Mike G. and Karel R., (1999), History of SSADM, SSADM an Introduction. pp. 2-5.